Graph Explorer: Training Faithful KG Agents with Visibility-Grounded Supervision

Yifeng Chen[1], Sicheng Wan[2], Tianyi Zhang[2], Xuezhou Zhang[1]
[1]Boston University [2]University of Washington, Seattle
chenyf08@bu.edu xuezhouz@bu.edu

Abstract

Large language models (LLMs) are strong reasoners but still hallucinate and make unreliable decisions on knowledge-intensive questions. Knowledge graphs (KGs) provide explicit, auditable facts, motivating KGQA agents that interact with KGs via tool calls to reduce hallucinations. However, LLM agents often struggle to reliably manipulate KG-specific symbols (entity IDs and relation names), leading to invalid or hallucinated tool-call arguments, and high-quality step-by-step supervision for such tool use is scarce. Meanwhile, large datasets of expert SPARQL programs exist for Freebase KGQA, but naively converting them into action supervision is brittle: SPARQL assumes a global view of the KG, while an agent acts from a truncated, local prompt, so expert steps can reference KG IDs (entity/relation/attribute symbols) that are not visible at decision time. We present Graph Explorer, a fully automatic data synthesis pipeline that turns expert SPARQL into executable, visibility-grounded (actions may use only IDs shown in the prompt) tool supervision without manual trace labeling. Graph Explorer compiles SPARQL into tool-call plans, executes them under the same context-control policy used at inference, and retains only tool-interaction traces whose tool-call arguments are visible at decision time, yielding clean (context, next-action) pairs for action-centric fine-tuning. We evaluate with a strict finish-or-fail protocol (success only if the agent issues a valid `Finish` within budget). Under this protocol, our fine-tuned Qwen3-8B reaches 74.0/80.2 Hit@1 on CWQ/WebQSP, improving over a reproduced prompting baseline by +22.5/+16.2 points, indicating more faithful multi-step graph exploration from visible evidence.

1 Introduction

Large language models (LLMs) have become strong general-purpose reasoners, and recent prompting and tool-use paradigms further extend them from pure text generation to interactive decision making (Wei et al., 2022; Yao et al., 2022; Schick et al., 2023; Wang et al., 2023b; Shinn et al., 2023; Zhong et al., 2024). However, relying solely on parametric knowledge can be brittle for knowledge-intensive questions: models may be out-of-date and can hallucinate unsupported facts (Lewis et al., 2020; Ji et al., 2022; Shuster et al., 2021). Knowledge graphs (KGs), which store facts as explicit entities and relations, provide a natural substrate for grounding and auditing such reasoning. This motivates knowledge-graph question answering (KGQA), where a model must answer natural language questions by retrieving and reasoning over a KG—often through a sequence of graph exploration actions. Recent research in KGQA has shifted from monolithic semantic parsing (Yih et al., 2016; Talmor and Berant, 2018) to agentic tool use (Sun et al., 2024; Chen et al., 2024; Xiong et al., 2024), where models answer questions through step-by-step graph exploration. A major bottleneck for these agents is the scarcity of supervision (Lan et al., 2023; Pan et al., 2023). Human annotation of reasoning traces is prohibitively expensive (Finegan-Dollak et al., 2018; Qiu et al., 2020; He et al., 2021), leading most existing methods to rely on weaker signals like prompting (Sun et al., 2024; Chen et al., 2024; Xiong et al., 2024). Although large datasets of expert SPARQL queries exist (Yih et al., 2016; Talmor and Berant, 2018), they have not been effectively repurposed to train agents because of a fundamental difference in how the two paradigms access the graph.

Semantic parsers operate with a global view of the KG, whereas agents act under a context-limited local view. Naively translating SPARQL into action sequences can require identifiers that are not visible to the agent, yielding ungrounded supervision that encourages hallucinated arguments rather than evidence-based reasoning.

To address this, we propose a method to con-

Findings of the Association for Computational Linguistics: ACL 2026, pages 7875–7890

vert SPARQL queries into reliable, grounded agent supervision. We first compile expert queries into linear action plans (i.e., a step-by-step sequence of JSON tool calls over our fixed action interface). Crucially, we screen out compiled steps that are not executable under the agent's limited observation, so the supervision never requires predicting identifiers the agent could not have seen.

We demonstrate the effectiveness of this approach on the ComplexWebQuestions (CWQ) benchmark (Talmor and Berant, 2018). We evaluate under a stricter *finish-or-fail* protocol. Prior KG agents often use a best-effort setting that allows a final guess when the step budget is exhausted, which can blur tool-grounded reasoning with parametric recall. We therefore report both BE (best-effort) for comparability and a strict FoF (finish-or-fail) protocol that only counts answers produced by a valid `Finish` within budget. Under FoF, our supervised agent remains robust while prompting baselines degrade substantially.

Contribution Highlights

- **Unlocking Data:** We propose a framework to repurpose existing expert logical queries into grounded agent supervision. By resolving the conflict between global queries and local agent views, we generate large-scale, high-quality training data without human annotation.
- **Faithful Agent:** We train a faithful, evidence-grounded agent that reasons only from model-visible evidence during graph exploration.
- **Rigorous Evaluation:** We introduce a strict "finish-or-fail" evaluation protocol. Unlike standard "best-effort" metrics that allow guessing, this protocol explicitly disentangles answers produced by valid tool-based evidence chains (`Finish` within budget) from those that could be explained by parametric recall, luck, or end-of-budget guessing.

2 Related Work

KGQA supervision: semantic parsing and weakly supervised agents. Classic KGQA commonly uses semantic parsing to map questions to executable logical forms (e.g., SPARQL) (Yih et al., 2015, 2016; Talmor and Berant, 2018). While logical forms provide explicit, verifiable reasoning, collecting question–program supervision is expensive and hard to scale across domains (Finegan-Dollak et al., 2018). To reduce annotation cost, a long line of work trains KG reasoning models with weak supervision from final answers (often via RL-style path search) (Das et al., 2018; Liang et al., 2017; Zhang et al., 2018; Qiu et al., 2020), but learning becomes challenging due to sparse/delayed rewards and spurious trajectories (Qiu et al., 2020). Several methods add intermediate signals or teacher guidance to stabilize learning under weak supervision (He et al., 2021), yet obtaining reliable step-level supervision for multi-hop decision making remains a bottleneck.

LLM-based KG agents and the supervision scarcity gap. Recent KG-augmented LLM frameworks treat the LLM as an agent that iteratively explores the graph: Think-on-Graph (ToG) (Sun et al., 2024) and Plan-on-Graph (PoG) (Chen et al., 2024) improve multi-hop performance through structured prompting and adaptive exploration; Interactive-KBQA (Xiong et al., 2024) uses multi-turn interactions to produce executable actions; KG-Agent (Jiang et al., 2024) learns a tool-using policy from synthesized stepwise supervision over KG reasoning data. Its interaction/termination semantics differ from our budgeted BE/FoF setting, so we do not treat the reported numbers as directly comparable. Despite strong results, many of these systems are still primarily prompt-driven (or rely on limited manual design), leaving a central gap: how to obtain scalable, executable step supervision for KG agents without expensive human traces. In this work, we address this gap by automatically repurposing existing expert SPARQL into agent trajectories while enforcing a visibility/observability constraint, ensuring that supervised actions are grounded in the agent's model-visible context rather than requiring prediction of hidden identifiers.

3 Problem Formulation

Given a natural language question q and a Knowledge Graph $\mathcal{G}$, the goal of KGQA is to retrieve a set of answer entities $\mathcal{A}_q \subseteq \mathcal{E}$. We formulate this task as a sequential decision-making process under partial observability.

Agent as a Partially Observable MDP. We define the agent's interaction as a Markov Decision Process (MDP) tuple $(\mathcal{S}, \mathcal{U}, \mathcal{T}, \mathcal{O})$ (Kaelbling et al., 1998). At step t, the environment maintains a latent

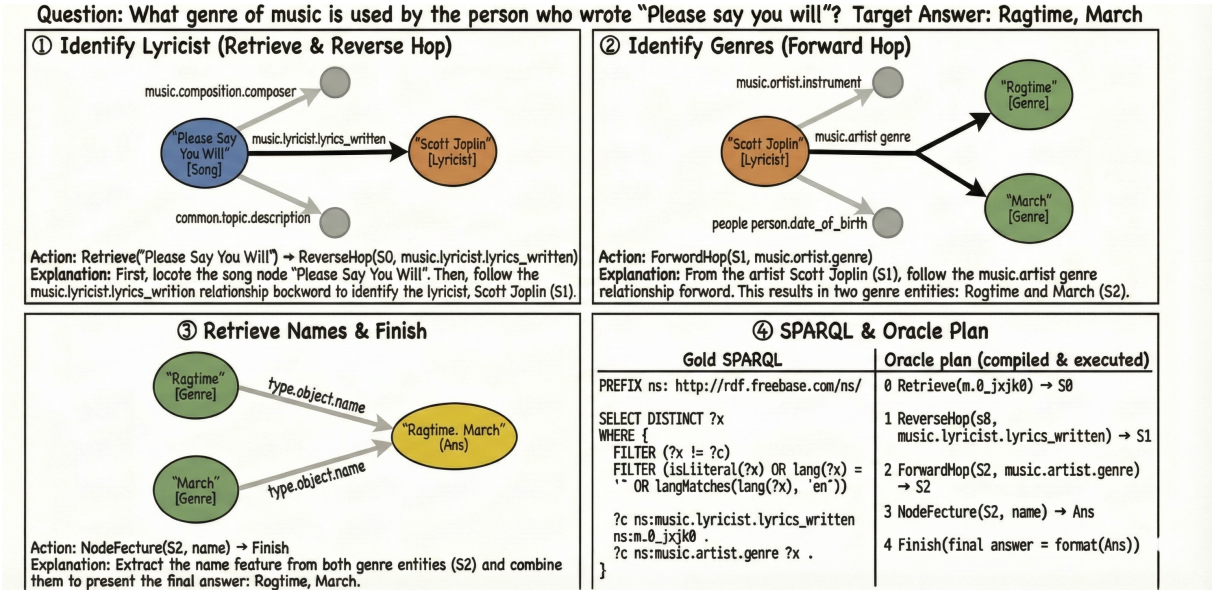

Figure 1: Running example on ComplexWebQuestions (CWQ). Left: the evidence chain realized by a sequence of tool calls (retrieve → hop → extract → finish). Right: Gold SPARQL (global view) and the corresponding oracle tool plan compiled under our fixed interface, where intermediate results are stored as registry-backed sets (`S0`–`S2`).

state $s_t \in \mathcal{S}$ (e.g., the full execution history and intermediate retrieval results). The agent issues an action $u_t \in \mathcal{U}$, and the environment transitions to s_{t+1} and emits an observation $o_t \in \mathcal{O}$ according to the transition function $(s_{t+1}, o_t) = \mathcal{T}(s_t, u_t)$. The interaction history up to step t is denoted as $h_t = (q, u_1, o_1, \ldots, u_t, o_t)$. A critical challenge in agentic KGQA is the resulting partial observability: the agent cannot condition on the full history h_t due to context window limits and the massive size of graph neighborhoods. We formalize this by introducing a deterministic context compression function ϕ. We refer to the model-visible prompt before choosing u_t as the *decision-time context* x_{t-1}. The agent's policy π_θ selects actions based only on this restricted, visible context:

$$x_{t-1} = \phi(h_{t-1}), \quad u_t \sim \pi_\theta(u_t \mid x_{t-1}) \quad (1)$$

where x_{t-1} is a deterministic, context-controlled serialization of h_{t-1}. We instantiate ϕ with a concrete prompt construction and context-control rules in Sec. 4.1.

Objective: Finish-or-Fail. Classic KGQA evaluates answers with Hit@1, i.e., whether the final predicted answer matches the expert answer set. In agentic KGQA, prior work often uses a best-effort variant (Sun et al., 2024; Chen et al., 2024): if the step budgets are exhausted before `Finish`, a final free-form guess is forced and scored with Hit@1, which can conflate tool-based reasoning with parametric knowledge. To isolate executable, evidence-grounded tool use, we adopt a strict finish-or-fail objective under budgeted interaction.

We impose a strict budgeted interaction setting to constrain both search depth and total latency. We define two budget limits: (1) **Hop Budget** (B): the maximum number of graph traversal actions (e.g., `ForwardHop`, `ReverseHop`) allowed in a trajectory. (2) **Total Action Budget** (T): the maximum number of total tool actions (including retrieval, set operations, and `Finish`). The agent succeeds if and only if it generates a correct `Finish` action without violating either budget ($|\tau_{\text{hops}}| \leq B$ and $|\tau_{\text{total}}| \leq T$). Formally, we aim to learn a policy π_θ that maximizes:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\Big[\mathbf{1}\{\text{Succ}(\tau; q, B, T)\}\Big]. \quad (2)$$

where $\text{Succ}(\tau; q, B, T)$ holds iff $\text{ans}(\tau) \in \mathcal{A}_q$, $|\tau_{\text{hops}}| \leq B$, and $|\tau_{\text{total}}| \leq T$. τ_{hops} is the subsequence of hop actions (`ForwardHop`/`ReverseHop`) in the trajectory, and τ_{total} counts all tool actions (including Retrieve / set ops / NodeFeature / Finish). This objective necessitates that the agent learns to prioritize visible evidence within x_{t-1} rather than relying on parametric memorization, as blind guesses are strictly penalized.

4 Method: Visibility-Grounded Supervision

4.1 Context Construction and Control

```
Q: Where are the Giza Pyramids located that appointed Hosni Mubarak to a governmental position?

(node tools take MIDs; set ops take set IDs; each call creates a fresh Si)

Compressed obs (t=0..3):
t=0  Thought: resolve the Giza Pyramids entity from the query mention.
     Action : S0 := RetrieveNode("Nekropole von Gizeh")  [Obs=c5832]

t=1  Thought: expand the Giza entity to its containing locations (candidates).
     Action : S1 := ForwardHop(src=[m.07s6gb8] from S0, rel=location.location.containedby) |S1|=2  [Obs=c5833]

t=2  Thought: resolve the appointer entity (Hosni Mubarak).
     Action : S2 := RetrieveNode("Hosni Mubarak")  [Obs=c5834]

t=3  Thought: fetch positions appointed by Hosni Mubarak (appointees / held positions).
     Action : S3 := ForwardHop(src=[m.0dnps] from S2, rel=government.political_appointer.appointees) |S3|=2  [Obs=c5835]


Raw obs (last W=2):
t=4  Thought: map the appointed positions to their jurisdiction (where the office applies).
             Action : S4 := ForwardHop(src=[m.0g93h27] from S3,
             ↪  rel=government.government_position_held.jurisdiction_of_office) |S4|=1
             Obs    : sample: [m.02k54] name="Egitto" (type=base.aareas.schema.earth.sovereign_state)
                      relation: location.country.capital; location.country.form_of_government;
                                government.governmental_jurisdiction.governing_officials; ...
                      feature : name; iso_code; official_name

t=5  Thought: join the Giza-location candidates with the jurisdiction candidates via intersection.
     Action : S5 := Intersect(sets=[S1,S4]) |S5|=1  out: [m.02k54] "Egitto"  (|S1|=2, |S4|=1)

Registry: S1=2; S4=1; S5=1

Thought: extract the final surface name for the remaining entity.
Next: NodeFeature(ids=members(S5), attr=name) -> "Egitto"
```

Figure 2: Decision-time prompt x_{t-1} at step t: the full action trace is visible, while observations are window-compressed (W=2) and older blocks are replaced by placeholders like `[Obs_ID=...]`. Each tool call creates an intermediate set stored in a latent registry and exposes only its handle S_i (e.g., `S1`, `S4`), so later set operators (e.g., `Intersect([S1,S4])`) remain executable under bounded context.

Overview. Freebase is large, and even a single tool call may return long lists of entities, relations, and facts, quickly exceeding an LLM's context window and breaking multi-step execution. To make multi-step reasoning feasible, we define an explicit agent–environment contract: (i) a fixed JSON tool interface whose actions operate over entity IDs (MIDs) and set handles (S_i, IDs for intermediate entity sets), producing intermediate sets, and (ii) a deterministic context construction ϕ that maps the interaction history to the model-visible prompt x_t (Wang et al., 2023b). Full intermediate results are stored in a latent set registry, functioning as an external memory, while x_t exposes only compact set handles and bounded previews of the latest tool outputs (Zhong et al., 2024).

Contract: tools, set handles, and what the model sees. The agent interacts with the KG via a small set of JSON tools that produce and consume sets of Freebase IDs. Traversal tools retrieve and expand entity sets (`RetrieveNode`; `ForwardHop`/`ReverseHop`). Set operators (`Intersect`/`Union`/`Diff`/`Filter`/`OrderBy`/`TopK`) combine and refine intermediate results using only set handles. Extraction tools (`NodeFeature`; `Finish`) read out literals (e.g., names) and terminate with the final answer.

To support long tool traces under a bounded context window, the environment maintains a latent set registry storing the full members of each intermediate set. Each produced set is assigned a handle S_i; the prompt x_t contains compact registry lines (e.g., "S_i := creator | size") and bounded previews, while subsequent steps refer to sets only by handles (e.g., `Intersect([S_2,S_4])`).

We distinguish entity identifiers (MIDs, e.g., `m.xxx`) from set handles (S_i): traversal tools consume MIDs (argument `src`) and produce a fresh set handle; extraction tools consume MIDs (argument `ids`); set operators consume set handles (e.g., `sets`/`from_set`).

The model-visible context is constructed as

$$x_t = [H;\ q;\ T_{<t};\ \tilde{O}_{<t};\ R_t].$$

Components. H is the static header (schema + one-shot demo); q is the question; $T_{<t}$ is the full thought/action trace up to step t–1 (uncompressed); $\tilde{O}_{<t}$ is the serialized observation history after context control (older blocks may be placeholders); R_t is the set-registry summary at step t (compact handle lines). We use x_{t-1} to denote the decision-time prompt before choosing u_t; it contains the trace

and observations up to step $t-1$. We keep the full action trace visible for executability, while observation content may be compressed as the trajectory grows (Sec. 4.1; full templates and exact caps in the appendix).

Deterministic context control (ϕ). We apply three deterministic rules to keep x_t within the context window while preserving grounding and executability. (1) *Windowed observation compression* (default $W=2$): only the most recent W observation blocks are kept verbatim; older ones are replaced by placeholders like `[Obs_ID=c]`, while the full action trace remains intact. (2) *Per-step display caps*: each tool response is rendered as a bounded preview by capping samples and local structure shown per node (e.g., candidate relations/facts), preventing high-degree neighborhoods from dominating the prompt; full contents remain latent in the registry. (3) *Token-level truncation*: if needed, a deterministic last-resort truncation is applied (details in the appendix).

Even though older observations are not verbatim in-context, the agent can still execute the next step because it relies on currently visible previews plus persistent set handles.

4.2 Bridging the Supervision Gap via Visibility Alignment

Under the agent interface and context-control rules in Sec. 4.1, we generate training data in two steps: (1) compile gold SPARQL into linear tool plans over our `JSON` operators (Sec. 4.2.1), and (2) execute these plans and keep only rollouts whose action arguments are visible in the model-visible context x_{t-1} (Visibility Check; Sec. 4.2.2), yielding (x_{t-1}, u_t) pairs for action-centric SFT. This trains a policy that selects grounded, executable actions based on visible evidence.

4.2.1 Symbolic Alignment: Compiling SPARQL to Linear Plans

A SPARQL query is a single global program that implicitly composes multiple operators—joins over KG triples (head, relation, tail), filters, set constraints (e.g., `EXISTS/NOT EXISTS`), and ranking/selection (`ORDER BY/LIMIT`)—under an omniscient view of the KG. In contrast, our agent operates under a fixed tool interface and can only act one executable tool call at a time, producing intermediate results as registry-backed sets. Therefore, SPARQL is not directly aligned with our action space: to obtain step-level supervision and executable traces, we must decompose each expert SPARQL query into a step-by-step sequence of tool calls, where later steps consume only intermediate sets/anchors created by earlier steps. We refer to this executable sequence as a linear tool plan (Yih et al., 2015). Full compilation rules and coverage are provided in Appendix F.

4.2.2 Execution Alignment: Visibility-Constrained Roll-out

For each question q, we execute the compiled linear tool plan in a Freebase-backed environment to obtain a step-by-step trajectory of (x_{t-1}, u_t, o_t), where x_{t-1} is the model-visible prompt at step t, u_t is the corresponding tool call, and o_t is the environment observation.

However, even after compilation, the plan is still created with a full-graph view. A step can be globally correct but refer to an identifier that is not retrievable from the model-visible prompt x_{t-1}: the agent may not have retrieved it yet, or it is omitted by context control (per-step observation display limits, windowed compression, or token-level truncation; Sec. 4.1).

We therefore require that all identifiers referenced by u_t be visible in x_{t-1}.

A supervised pair (x_{t-1}, u_t) is visibility-grounded if every identifier referenced by u_t (entity MID / relation / attribute / set id) occurs verbatim in the final model-visible context x_{t-1} (as defined by the compression strategy in Sec. 4.1), i.e., after observation compression and any token-level truncation.

In constructing step-level SFT data, we check this for every step (e.g., for `ForwardHop(src=a, rel=b)` or `ReverseHop(src=a, rel=b)`, both a and b must appear in x_{t-1}). If any step in a trajectory violates the constraint, we discard the entire trajectory to avoid training on actions that depend on hidden information. We refer to this trajectory-level pruning rule as the Visibility Check: an oracle rollout is kept if every step is visibility-grounded under the same context-control rules.

We apply it only when building SFT data and do not discard instances at evaluation time. This yields an SFT dataset $\mathcal{D} = \{(x_{t-1}, u_t)\}$ by extracting all step pairs from each retained oracle rollout, which we use for policy learning in Sec. 4.3.

Stage	CWQ		WebQSP	
	#Q	%	#Q	%
All questions	27,734	100.0	3,098	100.0
Compiled to linear tool plans	27,679	99.8	3,092	99.8
Pass Visibility Check (default limits)	20,643	74.4	2,623	84.6

Table 1: Coverage of plan compilation and Visibility Check filtering on CWQ and WebQSP. We compile expert SPARQL into linear tool plans and then execute oracle plans under the same tool interface. We retain only questions that pass a strict Visibility Check under the default KG expansion limits. An ablation that loosens expansion limits substantially increases the pass rate but yields impractically long observations; details are reported in Appendix G. We report the coverage and distributional impact of this filtering in Appendix K.

4.3 Policy Learning via Action-Centric Supervision

We fine-tune the agent policy π_θ using the visibility-grounded dataset $\mathcal{D}$ constructed in Sec. 4.2. Formally, for each training pair $(x_{t-1}, u_t) \in \mathcal{D}$, we optimize the standard next-token prediction objective, minimizing the negative log-likelihood of the action tokens conditioned on the visible history:

$$L(\theta) = -\mathbb{E}_{(x_{t-1},u_t)\sim\mathcal{D}}\Big[\log \pi_\theta(u_t \mid x_{t-1})\Big]. \quad (3)$$

where $\pi_\theta(u_t \mid x_{t-1})$ factorizes autoregressively as

$$\pi_\theta(u_t \mid x_{t-1}) = \prod_{j=1}^{|u_t|} \pi_\theta(u_{t,j} \mid x_{t-1}, u_{t,<j}),$$

and u_t is the target JSON action string (tokenized). We fine-tune Qwen3-8B (Qwen Team, 2025) using QLoRA (Dettmers et al., 2023); hyperparameters are in Appendix D.

Action-Only Supervision. A critical design choice is how to handle the `Thought` component of the ReAct paradigm (Yao et al., 2022) during training. Expert SPARQL queries do not contain natural language reasoning steps. While one could synthesize reasoning traces (e.g., via GPT-4), supervising the model on such synthetic thoughts risks inducing *hallucination* or *spurious correlations*, where the agent learns to mimic the style of the thought rather than grounding its decision in the observation x_{t-1}.

To enforce strict **state-dependency**, we adopt an *action-only supervision* strategy. We retain the ReAct format structure but populate the `Thought` field with a static placeholder in the training data. Crucially, we apply loss masking to the entire prompt (including the static thought) and compute gradients *only* on the JSON action tokens. This forces the model to bypass the lack of explicit reasoning supervision and learn to map the compressed observation history (in x_{t-1}) directly to the correct executable tool call, effectively distilling the expert's logical plan into the agent's policy.

5 Experiments

5.1 Experimental Setups

We evaluate on two Freebase KGQA benchmarks with expert SPARQL: ComplexWebQuestions (CWQ; Talmor and Berant, 2018) and WebQSP (Yih et al., 2016). Data preparation is described in Sec. 4.2.

Evaluation split. For efficiency and reproducibility, we report results on fixed subsets of 1,000 examples sampled with a fixed random seed 42 from CWQ-dev and WebQSP-test, and we release the corresponding QID lists. (The full CWQ-dev and WebQSP-test contain roughly ∼3k and ∼1.6k questions, respectively.)

Metric (Hit@1). Following common practice for Freebase KGQA (Sun et al., 2024; Chen et al., 2024), we report exact-match Hit@1 on the final predicted answer.

We additionally report a visibility-checked Hit@1 (VC) that discounts hits with any prompt-visibility violation (Visibility Check; Sec. 4.2.2).

Evaluation protocols. We report BE for cross-paper comparability (Table 2). Unless stated otherwise, we report FoF for tool-mastery evaluation under the budgets defined in Sec. 3. Scope convention. Tables 2–3 report results of the mixed-training model (CWQ+WebQSP), while the remaining analyses in Secs. 5.3–5.5 focus on CWQ only (training and evaluation on the CWQ 1k subset).

5.2 Comparison Methods

We compare against representative KGQA baselines in three groups (Table 2): (i) LLM-only prompting (IO, CoT, Self-Consistency), (ii) fine-tuned KG-augmented systems (e.g., UniKGQA, DeCAF, RoG, StructGPT), and (iii) prompting-based KG agents (e.g., ToG, PoG, InteractiveKBQA). Baseline numbers in Table 2 for UniKGQA, TIARA, RE-KBQA, DeCAF, RoG, KD-CoT, KB-BINDER, StructGPT, ToG, InteractiveKBQA, and PoG are from the original papers (Jiang et al., 2022; Shu et al., 2022; Cao et al., 2023; Yu et al.,

2023; Luo et al., 2024; Wang et al., 2023a; Li et al., 2023; Jiang et al., 2023; Sun et al., 2024; Xiong et al., 2024; Chen et al., 2024). SC denotes Self-Consistency (Wang et al., 2023c). For cross-paper reference, Table 2 reports best-effort accuracy (Hit@1_{BE}). Under our FoF protocol (Sec. 5.1), Table 3 reports Hit@1_{FOF} under a hop budget B with two requirements: (i) a valid `Finish` must be produced within budget (no best-effort guessing), and (ii) the episode must execute at least one KG tool call before `Finish` (tool-required success). We reproduce PoG as the closest directly comparable prompting baseline. ToG is not directly comparable because it may terminate with a direct answer without any KG tool call, while Interactive-KBQA is not directly comparable because it outputs the final tool observation under `Done` (observation-as-output) rather than a budgeted `Finish` action.

5.3 Performance Comparison

Unless otherwise stated, Tables 2–3 use a single model fine-tuned on the mixed CWQ+WebQSP supervision and evaluated on the fixed 1k subsets described in Sec. 5.1. We first compare our method against representative KGQA baselines under the commonly used best-effort protocol to enable cross-paper comparability. Table 2 shows that supervised fine-tuning with our executable supervision yields a substantial improvement on CWQ: Qwen3-8B increases from 39.2 (no FT) to 78.4 Hit@1_{BE}, far above LLM-only prompting baselines (37.6–45.4) and also stronger than prior fine-tuned KG-augmented systems such as DeCAF (70.4) and RoG (62.6). Among prompting-based KG agents, our FT model outperforms PoG with Qwen3-8B (56.8) by a wide margin and is competitive even with GPT-4 prompting results on CWQ (e.g., PoG 75.0). On WebQSP, fine-tuning similarly improves accuracy from 63.1 to 86.1 Hit@1_{BE}, approaching the strongest GPT-4 prompting baseline (PoG 87.3) while exceeding most non–GPT-4 methods reported in Table 2.

We next evaluate under our strict finish-or-fail (FoF) protocol (Sec. 5.1), where an episode is scored correct only if it emits a well-formed `Finish` within the hop budget (no final free-form guess is allowed). Under FoF, the gains observed under best-effort persist: with $B = 8$, our SFT model achieves 74.0 Hit@1_{FoF} on CWQ, compared to 14.4 for the Qwen3 base model and 51.5 for reproduced PoG (Table 3), indicating that improvements are not explained by budget-exhaustion guessing. Notably, the BE→FoF drop is modest for our FT model (78.4→74.0) but severe for the base model (39.2→14.4), consistent with fine-tuning improving executable tool use and termination rather than relying on forced final guesses. A similar pattern holds on WebQSP: our model reaches 80.2 Hit@1_{FoF} (vs. 35.3 base and 64.0 reproduced PoG), while maintaining strong best-effort performance (86.1 Hit@1_{BE}).

Method	CWQ	WebQSP
LLM-Only		
IO Prompt	37.6	63.3
CoT	38.8	62.2
SC	45.4	61.1
Fine-Tuned KG-Augmented LLM		
UniKGQA	51.2	79.1
TIARA	–	75.2
RE-KBQA	50.3	74.6
DeCAF	70.4	82.1
RoG	62.6	85.7
Prompting KG-Augmented LLM w/ GPT-3.5 or others		
KD-CoT	50.5	73.7
KB-BINDER	–	74.4
StructGPT	54.3	72.6
ToG	57.1	76.2
PoG	63.2	82.0
Prompting KG-Augmented LLM w/ GPT-4		
InteractiveKBQA	59.2	72.5
ToG	67.6	82.6
PoG	75.0	87.3
Prompting KG-Augmented LLM w/ Qwen3-8B		
ToG	38.6	65.2
PoG	56.8	66.8
Qwen3-8B	39.2	63.1
ours Qwen3-8B FT	78.4	86.1

Table 2: Performance comparison on CWQ and WebQSP under **best-effort** termination (Hit@1_{BE}, %). Our prompting baselines and reproduced ToG and PoG are evaluated on the fixed 1k subsets; other numbers are taken from original papers (potentially with different evaluation settings) and shown for reference.

		Hit@1_{FoF} (%)	
Method	B	CWQ	WebQSP
PoG (Qwen3, reproduced)	8	51.5	64.0
ours Qwen3-8B FT	8	74.0 (73.1$^{\mathrm{VC}}$)	80.2 (79.8$^{\mathrm{VC}}$)
Qwen3-8B base	8	14.4	35.3

Table 3: Strict termination comparison on CWQ and WebQSP under hop budget B (Hit@1_{FoF}, %). $^{\mathrm{VC}}$ reports visibility-checked Hit@1 (Sec. 5.1) under the same context truncation strategy (Visibility Check; Sec. 4.2.2).

As defined in Sec. 5.1, visibility-checked Hit@1 (VC) discounts hits with prompt-visibility violations. For Qwen3-8B FT ($B = 8$), Hit@1_{FoF} is 74.0 on CWQ and only 0.9 points correspond to hits with evidence violations, yielding a VC of 73.1 (Table 3); on WebQSP the gap is similarly small (80.2 vs. 79.8 VC). This small FoF→VC delta suggests that most successful episodes are achieved without overreaching beyond prompt-visible identifiers, aligning with our goal of evidence-grounded, visibility-consistent action selection. Detailed analysis in Appendix J provides further details.

Takeaway. Executable, visibility-grounded step supervision yields large gains not only under best-effort scoring but also under strict termination, indicating improved evidence-grounded execution rather than budget-exhaustion guessing. We further analyze trajectory alignment and operator-level plan adherence; see Appendices H and I.

5.4 Ablation Study

In this section, we focus on CWQ only: all models are trained on CWQ filtered supervision and evaluated on the CWQ 1k subset.

Visibility Check. We ablate the Visibility Check used during SFT data construction (Sec. 4.2.2). Both results are from models trained on 8k trajectories; the only difference is whether the 8k supervision is filtered by the Visibility Check. As shown in Table 4, enabling the Visibility Check improves finish rate from 76.7% to 82.9% and increases Hit@1_{FoF} by +6.6 points (64.6 → 71.2). This suggests that the Visibility Check helps both reaching valid termination and improving correctness, consistent with more grounded and executable next-action learning. An ablation on the effect of training data size on performance is provided in Appendix E.

Setting	Finish	Hit@1_{FoF}
w/o Visibility Check	76.7	64.6
w/ Visibility Check	82.9	71.2
Δ (on - off)	+6.2	+6.6

Table 4: Ablation on Visibility Check under **finish-or-fail** evaluation ($N = 1000$ for both). Values are percentages.

Interaction Budgets. We study how strict performance varies with interaction budgets. Hit@1_{FoF} increases with hop budget B and saturates around $B \approx 7$–8 for SFT; the base model remains low even with larger budgets (Table 5), suggesting budget alone cannot compensate for weak grounded action selection. We additionally sweep the global action budget T; see Appendix L.

B	Qwen3-SFT	PoG	Qwen3-Base
1	4.3	36.9	5.2
2	31.2	44.5	11.6
3	58.3	48.3	13.1
4	68.6	50.1	14.2
5	71.0	51.2	14.3
7	73.6	51.5	14.4
8	73.7	51.5	14.4
15	73.7	51.5	14.4

Table 5: Hit@1_{FoF} under hop-budget truncation (**finish-or-fail**) for Qwen3-SFT, PoG, and Qwen3-Base.

Oracle-Relation Masking (Stress Test). We test whether the agent relies on prompt-visible evidence by removing, at inference time, the hop relations used in the compiled expert plan from the model-visible candidate relation lists, while keeping the backend KG unchanged. As shown in Table 6, Hit@1_{FoF} drops from 73.7% to 6.0%, indicating that correct answers rarely occur without the required prompt-visible hop evidence. This sharp drop suggests that the learned policy primarily follows prompt-visible hop evidence, rather than recovering missing relations from parametric memory.

Setting	Hit@1_{FoF} (%)
Standard CWQ-dev	73.7
Oracle-relation masked	6.0

Table 6: Oracle-relation masking stress test: we remove the expert-plan hop relations from prompt-visible candidates at inference time. Values are percentages.

6 Conclusion

We study tool-using KGQA under partial observability, where the agent only sees a context-controlled history. We propose visibility-aware supervision that filters training steps so action arguments are grounded in what the model can see. With a finish-or-fail protocol and execution-focused metrics, we evaluate agents beyond final answers. On CWQ, our approach significantly improves both accuracy and execution behavior; future work will relax expert-entity assumptions and extend grounding to attributes and noisier settings.

7 Limitations.

While our framework improves evidence-grounded tool use for KGQA agents, it has several limita-

tions. First, we primarily study a single open-source backbone (Qwen3-8B); more experiments on additional open models and sizes are needed to assess robustness and generality. Second, our evaluation focuses on SPARQL-annotated Freebase benchmarks (CWQ/WebQSP) and fixed 1k subsets; cross-domain generalization to other KGs and benchmarks (potentially with different query languages or program annotations) remains future work. Third, our pipeline assumes dataset-provided topic entities and executable expert queries; extending to settings with noisy entity linking, incomplete/missing program annotations, or other knowledge sources is an open direction. Overall, we view this work as a starting point toward scalable supervision for faithful KG agents.

Acknowledgments

We thank Lambda (https://lambda.ai) for cloud compute credits provided through the Lambda Research Grant Program. We used AI-based writing assistants (e.g., ChatGPT) only for minor language editing and polishing of author-written text, and used an AI image-generation system to create the illustration in Figure 1. All technical content, analyses, and conclusions were written and verified by the authors.

References

Yong Cao, Xianzhi Li, Huiwen Liu, Wen Dai, Shuai Chen, Bin Wang, Min Chen, and Daniel Hershcovich. 2023. Pay more attention to relation exploration for knowledge base question answering. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 2119–2136, Toronto, Canada. Association for Computational Linguistics.

Liyi Chen, Panrong Tong, Zhongming Jin, Ying Sun, Jieping Ye, and Hui Xiong. 2024. Plan-on-graph: Self-correcting adaptive planning of large language model on knowledge graphs. *Preprint*, arXiv:2410.23875.

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. 2018. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. arXiv preprint arXiv:2305.14314.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

Gaole He, Yunshi Lan, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. Improving multi-hop knowledge base question answering by learning intermediate supervision signals. In *Proceedings of the Fourteenth ACM International Conference on Web Search and Data Mining (WSDM)*, pages 553–561.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Delong Chen, Wenliang Dai, Ho Shu Chan, Andrea Madotto, and Pascale Fung. 2022. Survey of hallucination in natural language generation. *Preprint*, arXiv:2202.03629.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. StructGPT: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9237–9251.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, Yang Song, Chen Zhu, Hengshu Zhu, and Ji-Rong Wen. 2024. KG-Agent: An efficient autonomous agent framework for complex reasoning over knowledge graph. *Preprint*, arXiv:2402.11163.

Jinhao Jiang, Kun Zhou, Xin Zhao, and Ji-Rong Wen. 2022. Unikgqa: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. In *International Conference on Learning Representations (ICLR)*.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134.

Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji Rong Wen. 2023. Complex knowledge base question answering: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 35(11):11196–11215.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. 2023. Few-shot in-context learning on knowledge base question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6966–6980, Toronto, Canada. Association for Computational Linguistics.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23–33, Vancouver, Canada. Association for Computational Linguistics.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2024. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *International Conference on Learning Representations (ICLR)*.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2023. Unifying large language models and knowledge graphs: A roadmap. *arXiv preprint arXiv:2306.08302*.

Yunqi Qiu, Yuanzhuo Wang, Xiaolong Jin, and Kun Zhang. 2020. Stepwise reasoning for multi-relation question answering over knowledge graph with weak supervision. In *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3–7, 2020*, pages 474–482. ACM.

Qwen Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36.

Yiheng Shu, Zhiwei Yu, Yuhan Li, Börje F. Karlsson, Tingting Ma, Yuzhong Qu, and Chin-Yew Lin. 2022. TIARA: Multi-grained retrieval for robust question answering over large knowledge base. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8108–8121, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. 2021. Retrieval augmentation reduces hallucination in conversation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3784–3803, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M. Ni, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *International Conference on Learning Representations (ICLR)*.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 641–651.

Keheng Wang, Feiyu Duan, Sirui Wang, Peiguang Li, Yunsen Xian, Chuantao Yin, Wenge Rong, and Zhang Xiong. 2023a. Knowledge-driven cot: Exploring faithful reasoning in llms for knowledge-intensive question answering. *Preprint*, arXiv:2308.13259.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. 2023b. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023c. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837.

Guanming Xiong, Junwei Bao, and Wen Zhao. 2024. Interactive-KBQA: Multi-turn interactions for knowledge base question answering with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10561–10582, Bangkok, Thailand. Association for Computational Linguistics.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1321–1331, Beijing, China. Association for Computational Linguistics.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 201–206.

Donghan Yu, Sheng Zhang, Patrick Ng, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Yiqun Hu, William Yang Wang, Zhiguo Wang, and Bing Xiang. 2023. Decaf: Joint decoding of answers and logical forms for question answering over knowledge bases. In *International Conference on Learning Representations (ICLR)*.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J. Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 6069–6076.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, Ye He, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19724–19731.

A System prompt and output contract

We provide the full system prompt and the action-only output contract used by GraphAgent. For readability, we omit implementation-specific wrappers and show the exact text presented to the LLM.

```
You are GraphAgent for Freebase.

Goal: Answer by issuing graph actions. Loop:
↪  Thought -> Action -> Observation. When
↪  certain, call Finish.

Actions (one-line JSON):
- Node: {"name":"RetrieveNode","args":{"keyword
↪  ":"<text-or-mid>"}}
        {"name":"ForwardHop","args":{"src":["<m
        ↪  id>"],"rel":"domain.type.property",
        ↪  "k":1}}
        {"name":"ReverseHop","args":{"src":["<m
        ↪  id>"],"rel":"domain.type.property",
        ↪  "k":1}}
       {"name":"NodeFeature","args":{"ids":["<mi
        ↪  d>"],"attr":"domain.type.property"}}
- Set:  {"name":"Filter","args":{"from_set":"<s
↪  et-id>","attr":"<attr>","op":"==","value":"
↪  <value>"}}
        {"name":"OrderBy","args":{"from_set":"<
        ↪  set-id>","attr":"<attr>","dir":"ASC
        ↪  |DESC"}}
        {"name":"TopK","args":{"from_set":"<set
        ↪  -id>","k":1}}
        {"name":"Union","args":{"sets":["<set-i
        ↪  d-a>","<set-id-b>"]}}
        {"name":"Intersect","args":{"sets":["<s
        ↪  et-id-a>","<set-id-b>"]}}
        {"name":"Diff","args":{"sets":["<set-id
        ↪  -a>","<set-id-b>"]}}
- Final: {"name":"Finish","args":{"final
↪  answer":"<text>"}}

Type discipline:
- src/ids: mids only (m.xxx), never set IDs.
- from_set/sets: set IDs only (S0, S1, ...),
↪  never mids.
- Use only set IDs in the Set Registry; ignore
↪  *_helper sets; do not invent set IDs.

Sets:
- RetrieveNode/hops create new sets. Registry
↪  shows origin/size.
- Set ops
↪  (Filter/OrderBy/TopK/Union/Intersect/Diff)
↪  only use set IDs.
- To read features: narrow to a small set -> read
↪  mids from Observation -> call NodeFeature on
↪  those mids.

Workflow:
- Always start with RetrieveNode.
- If the prompt includes a [starting id] list,
↪  pick the MID (m.xxx) from that list
  as the RetrieveNode keyword (do not include
  ↪  extra text).
- Then ForwardHop/ReverseHop to follow relations;
↪  use OrderBy+TopK or Filter on sets when
↪  ranking.
- When answer is known, call Finish with a short
↪  text.
- Thought: one plain sentence. Action:
↪  single-line JSON.
```

B One-shot demonstration

We include one in-context example used in the fixed header.

```
### Example QA: Westminster -> country ###

Question:
Which country has the capital Westminster?

Thought:
Retrieve Westminster to ground the location.

Action:
{"name":"RetrieveNode","args":{"keyword":"Westm
↪  inster"}}

Observation:
S0 := Retrieve("Westminster") | size=1
  [m.09bkv] node_type=City name="Westminster"
  relations=[location.country.capital(1)]

Thought:
Follow location.country.capital in reverse to
↪  get the country.

Action:
{"name":"ReverseHop","args":{"src":["m.09bkv"],
↪  "rel":"location.country.capital","k":1}}

Observation:
S1 := ReverseHop(S0, location.country.capital) |
↪  size=1
  [m.024pcx] node_type=Country name="Korolevstvo
  ↪  Angliya"

Thought:
We already see the country node and its name.
Answer with that country and call Finish.

Action:
{"name":"Finish","args":{"final
↪  answer":"Korolevstvo Angliya"}}
```

Question:	What genre of music is sued by the person who wrote the song "Please say you will"?
Final answer	Ragtime; March.
Step 0	`RetrieveNode("Please Say You Will")` $\rightarrow S_0$ (l.l=1): `m.0_jxjk0` (`"Please Say You Will"`). **Visible relations (subset):** `music.composition.lyricist`(1) (chosen), `music.composition.composer`(1), `music.composition.recordings`(4), `music.composition.form`(1), `music.recording.song`(4).
Step 1	`ForwardHop(src=[m.0_jxjk0], rel=music.composition.lyricist, k=1)` $\rightarrow S_1$ (l.l=1): `m.07b69` (`"Scott Joplin"`). **Visible relations (subset):** `music.artist.genre` (chosen), `music.artist.album`, `music.artist.origin`, `music.composer.compositions`, `music.lyricist.lyrics_written`.
Step 2	`ForwardHop(src=[m.07b69], rel=music.artist.genre, k=1)` $\rightarrow S_2$ (l.l=2): `m.06m6j` (`"Ragtime"`), `m.0cdr_y` (`"March"`).
Step 3	`NodeFeature(ids=[m.06m6j, m.0cdr_y], attr=name)` $\rightarrow$ names: Ragtime; March. `Finish("Ragtime, March")`.
Grounded execution	Each hop source MID is taken from the immediately preceding observation set (e.g., `m.0_jxjk0` from S_0 and `m.07b69` from S_1), so the trajectory satisfies the execution-time grounding constraint that action starting points must be visible in x_{t-1} under the same context truncation strategy.

Table 7: Case study trace with a readable subset of visible relations (chosen relations marked).

C Case Study: Sued-by genre question

QID: `q_what_genre_of_music_is_sued_by_the_person_who_`

D Fine-tuning Details

We fine-tune `Qwen/Qwen3-8B` using QLoRA SFT. We keep the *effective batch size per optimizer update* constant across runs:

$$B_{\text{eff}} = (\#\text{GPUs}) \times (\text{batch per GPU}) \times (\text{grad\_accum}). \quad (4)$$

For example, $4\times1\times12$ and $8\times1\times6$ both yield $B_{\text{eff}} = 48$.

Command (8 GPUs).

```
CUDA_VISIBLE_DEVICES=0,1,2,3,4,5,6,7 \
torchrun --nproc_per_node=8 \
  -m tot.sft.train_qlora_sft \
  --model Qwen/Qwen3-8B \
  --data \
   data/merged/cwq_test_002.chat.jsonl \
  --out \
    runs/cwq_qwen3_lora_r96_len16k \
  --epochs 1 --batch 1 \
  --grad_accum 6 \
  --max_seq_len 16000 \
```

Setting	Value
Backbone model	`Qwen/Qwen3-8B`
Training objective	Supervised fine-tuning (SFT)
Parameter-efficient tuning	QLoRA
LoRA rank (r)	96
LoRA alpha	384
LoRA dropout	0.05
Max sequence length	16,000
Learning rate	1×10^{-4}
Epochs	1
Batch size (per GPU)	1
Gradient accumulation	6 (8 GPUs) / 12 (4 GPUs)
Effective batch per update	48

Table 8: QLoRA SFT hyperparameters.

Traj. passing Visibility Check	Epochs	Eff. batch	Approx. updates/epoch	Hit@1_{FoF}
8k	1	48	167	71.2
20,643	1	48	430	73.6

Table 9: Effect of Visibility Check–passed supervision budget. Both runs use the same hyperparameters and are trained for one epoch; hence larger supervision implies more optimizer updates.

```
  --lr 1e-4 \
  --lora_r 96 \
  --lora_alpha 384 \
  --lora_dropout 0.05
```

E Ablation: Visibility Check–Passed Supervision Budget

We ablate the amount of Visibility Check–passed supervision by training on a random subset of 8k trajectories from the Visibility Check–passed set (20,643). Both settings use identical hyperparameters and are trained for one epoch; therefore, the larger Visibility Check–passed set results in more optimizer updates. As shown in Table 9, increasing the Visibility Check–passed supervision budget (and thus optimization updates) yields a modest improvement of +2.4 Hit@1_{FoF}.

F SPARQL-to-Linear-Plan Compiler (`sql2plan`)

Goal and outputs. Given a Freebase SPARQL query (CWQ/WebQSP/GraphQuestions), `sql2plan` deterministically produces a linear executable plan $\pi = (z_1, \ldots, z_L)$ over our tool primitives. For debugging, we also output a native linearization that preserves the original SPARQL structure. Each compiled instance includes: (i) a parsed SPARQL AST, (ii) native linear plan text, (iii) visible tool-plan text, and (iv) structured steps with `op/args/requires/produces` for alignment with runtime execution traces.

SPARQL pattern	Compiled visible steps (sketch)
`ns:m.* p ?x`	RETRIEVENODE(`m.*`) → HOP(p, fwd)
`?x p ns:m.*`	reverse traversal from `m.*` or constrain current `?x`
`?x p ?y`	HOP(p, fwd/rev) once one endpoint is bound
`FILTER(?v = "s")`	FILTER(`attr`, `==`, `s`) (optionally hop-to-owner and hop-back)
`FILTER EXISTS {...}`	INTERSECT(`main`, `sub`)
`FILTER NOT EXISTS {...}`	DIFF(`main`, `sub`)
`ORDER BY ... LIMIT k`	ORDERBY + TOPK(k) (supports cross-hop keys via BFS)
Property path `p1/p2` or `^p`	statically expanded into a triple chain before compilation

Table 10: Representative compilation rules from SPARQL to the visible linear tool plan.

Parsing and normalization. We parse SELECT variables, WHERE triples, FILTER blocks (including EXISTS/NOT EXISTS), ORDER BY keys, LIMIT, and top-level UNION. We support Freebase constants of the form `ns:m.*` and `ns:g.*`. We statically expand the two common CWQ property-path forms: concatenation `p1/p2/...` into a triple chain with fresh intermediate variables, and inverse `^` into a reversed edge. Property paths containing $|$, $?$, $*$, or $+$ are treated as out-of-scope and gated.

Visible linear plan generation. We compile the normalized AST into a sequence of tool steps while maintaining (a) a constant pool `ns:m.*` $\rightarrow S_i$ and (b) a variable-to-set map `?v` $\rightarrow S_i$. Each step produces a set id (except NODEFEATURE) and declares dependencies via `requires`. Key compilation rules include: (i) anchor branches from `ns:m.* p ?v`, (ii) UNION compiled per-branch then merged by UNION, (iii) EXISTS/NOT EXISTS compiled as INTERSECT/DIFF, (iv) ORDER BY + LIMIT compiled as ORDERBY + TOPK (with BFS to reach cross-hop keys when needed), and (v) common temporal overlap patterns compiled into a single FILTER rendered as `overlap(from,to,[L,R])`.

Coverage on CWQ. On CWQ, the compiler produces non-gated linear tool plans for 27,679/27,734 questions (99.8%). The remaining failures are dominated by out-of-subset surface forms or explicitly gated property-path operators.

Why Visibility Check retains 74.4%. After compilation, we execute oracle plans under the same tool interface and apply a strict Visibility Check: any identifier used by a step must be grounded in the model-visible context at that step. Under resource-bounded KG access (Appendix G), intermediate neighborhoods and predicate lists may be truncated, causing some required identifiers to be unavailable to the next oracle step. Because we discard an entire oracle trajectory if any step violates visibility, this yields a conservative but clean supervision subset: 20,643 questions (74.4% of all) pass the Visibility Check under the default limits.

G Effect of KG Expansion Limits on the Visibility Check

Default limits. Our default configuration caps KG expansion fan-out and predicate enumeration to control observation length and keep the training context within budget. We use the following endpoint/graph limits in all main experiments:

- `FREEBASE_ODBC_TIMEOUT` = 10
- `FREEBASE_HTTP_TIMEOUT` = 12
- `FREEBASE_HTTP_MAX_RETRY` = 0
- `FB_HOP_PER_STEP_LIMIT` = 500
- `FB_MAX_TRIPLES_PER_NODE` = 500
- `FB_LIST_PRED_LIMIT` = 2000

Loosening limits increases pass rate but breaks training feasibility. On a random sample of 1,000 questions, we loosen the three fan-out related caps (`FB_HOP_PER_STEP_LIMIT`, `FB_MAX_TRIPLES_PER_NODE`, `FB_LIST_PRED_LIMIT`) from 500/500/2000 to 20,000 (keeping other settings unchanged). This increases the Visibility Check pass rate to 99.0% (990/1,000). However, the resulting observations become substantially longer due to much larger intermediate neighborhoods and predicate lists, making the prompt/context too long for efficient training. Therefore, we use the default limits throughout.

H Hop-overlap Analysis

Hop-overlap (execution-oriented metric; evidence alignment) To quantify whether an agent retrieves the intended graph evidence under a hop budget, we compute a hop-overlap score against the reference plan. We extract the ordered list of hop operations from the oracle plan (ignoring set operations/filters) and the hop operations actually executed by the agent. A hop is considered matched if it uses the same relation (and direction); the hop-overlap is the fraction of reference hops that are matched by the agent's executed hops.

Table 11 shows that SFT models achieve high hop-overlap for short plans (e.g., Qwen3-SFT overlap 100.0% at 1 hop and 94.76% at 2 hops),

plan_hops	n	Hit@1_{FoF} (%)	hop_overlap (%)
qwen2.5 sft			
1	66	75.76	100.00
2	314	77.07	92.36
3	337	79.53	91.49
4	179	63.69	87.29
5	71	54.93	84.51
6	18	16.67	76.85
7	4	0.00	35.71
Qwen3 base			
1	66	6.06	21.21
2	314	25.16	34.08
3	337	16.32	19.58
4	179	13.97	13.69
5	71	1.41	14.37
6	18	0.00	11.11
7	4	75.00	14.29
Qwen3 sft			
1	66	71.21	100.00
2	314	81.53	94.75
3	337	77.74	90.70
4	179	63.69	86.45
5	71	52.11	84.23
6	18	50.00	78.70
7	4	0.00	35.71

Table 11: Hit@1_{FoF} and hop overlap by reference plan hops (**finish-or-fail**). Values are percentages.

and the overlap gradually decreases as reference hop length increases, which co-occurs with decreasing Hit@1_{FoF} on longer-hop questions (e.g., `plan_hops` 4–5). This pattern is consistent with compounding retrieval/execution errors under strict budgets: errors in early hop selection make downstream steps less likely to align with the reference chain.

Model	plan_hops	Overlap tercile	overlap_avg (%)	Hit@1_{FoF} (%)
Qwen2.5-7B SFT	4	low	63.75	28.33
		mid	97.92	76.67
		high	100.00	85.00
	5	low	60.87	30.43
		mid	91.67	54.17
		high	100.00	79.17
Qwen3 SFT	4	low	61.67	36.67
		mid	97.92	75.00
		high	100.00	80.00
	5	low	64.35	21.74
		mid	87.50	50.00
		high	100.00	83.33
Qwen3 Base	2	low	0.00	4.76
		mid	35.71	22.86
		high	66.19	47.62
	3	low	0.00	8.85
		mid	15.93	11.50
		high	42.69	28.07

Table 12: Within each plan_hops bucket, higher hop overlap terciles correlate with higher Hit@1_{FoF} (**finish-or-fail**). Values are percentages.

Crucially, hop-overlap is also predictive of correctness within the same hop-length bucket. Table 12 stratifies examples by overlap terciles and shows a strong monotonic relationship: for Qwen3-SFT at `plan_hops=4`, accuracy rises from 36.67% (low overlap, 61.67%) to 80.00% (high overlap, 100%); a similar trend holds for `plan_hops=5` and for Qwen2.5-SFT. This supports the interpretation that gains from executable supervision reflect better evidence acquisition and relation/direction selection, rather than merely improved answer formatting.

I Additional Analysis: Strict Plan Prefix Completion

This appendix reports an operator-level plan adherence metric that goes beyond hop overlap. Given a compiled expert plan, we represent each plan step by a *signature* that captures the operator type and key arguments. In our setting, step signatures include: `Retrieve(mid)`, `Hop(rel,dir)`, and higher-level set/selection operators such as `Intersect`, `Diff`, `Filter`, `Sort`, `TopK`, and `NodeFeature`.

PlanMatch (order-agnostic coverage). Let the compiled plan be a sequence of step signatures $S = [s_1, \ldots, s_L]$. Let A^+ denote the set of *successfully executed* action signatures from the agent trajectory. We define plan matching coverage as:

$$\textsc{PlanMatch} = \frac{1}{L}\sum_{i=1}^{L} \mathbf{1}\big[s_i \in A^+\big]. \quad (5)$$

This metric measures whether the agent executed the operators required by the plan, ignoring the order.

StrictPrefixRatio (order-sensitive prefix completion). To measure whether the agent follows the plan *in order*, we compute the longest completed prefix of the plan. Formally, define the largest $m \in \{0, \ldots, L\}$ such that there exist indices $1 \le j_1 < j_2 < \cdots < j_m$ where each s_i is matched by a successful executed action signature at position j_i. The strict prefix completion ratio is:

$$\textsc{StrictPrefixRatio} = \frac{m}{L}. \quad (6)$$

Intuitively, a step is counted as completed only if *all previous steps are completed and the current step is completed as well*; out-of-order execution or skipping steps does *not* contribute to the prefix length.

Aggregation and missing samples. We report averages over `total_n=1000` questions for each run. If a sample is missing in the log directory, we assign its metrics to 0 (thus penalizing missing trajectories in the aggregate).

Results. Tables 13 and 14 report per-bucket results by oracle `plan_hops`. Overall, the SFT model achieves an average StrictPrefixRatio of 82.0% (loaded=993/1000), while the base model achieves 23.2% (loaded=989/1000), indicating that SFT substantially improves order-sensitive plan following, especially for plans with ≥ 3 hops.

Qwen3-8B SFT
(loaded=993/1000; avg StrictPrefixRatio=82.0%)

plan_hops	n	Hit@1	Hit@1|finish	PlanMatch	StrictPrefixRatio
1	66	71.2%	82.5%	92.1%	81.9%
2	315	81.3%	85.6%	94.4%	89.8%
3	339	77.3%	86.2%	90.4%	85.0%
4	180	63.9%	83.9%	83.7%	75.1%
5	71	52.1%	77.1%	79.3%	60.4%
6	18	50.0%	81.8%	74.3%	45.9%
7	4	0.0%	0.0%	36.2%	16.9%
missing	7	0.0%	0.0%	0.0%	0.0%

Table 13: Operator-level plan adherence and strict prefix completion by oracle `plan_hops` (total_n=1000; missing samples are assigned 0). PlanMatch is order-agnostic coverage; StrictPrefixRatio is order-sensitive prefix completion.

Qwen3-8B Base
(loaded=989/1000; avg StrictPrefixRatio=23.2%)

plan_hops	n	Hit@1	Hit@1|finish	PlanMatch	StrictPrefixRatio
1	63	6.4%	36.4%	32.1%	23.9%
2	315	25.1%	74.5%	35.8%	28.3%
3	340	16.2%	67.9%	26.5%	20.8%
4	178	14.0%	59.5%	20.8%	14.7%
5	71	1.4%	12.5%	19.0%	14.0%
6	18	0.0%	0.0%	17.5%	10.4%
7	4	75.0%	100.0%	14.9%	1.7%
missing	11	0.0%	0.0%	0.0%	0.0%

Table 14: Operator-level plan adherence and strict prefix completion by oracle `plan_hops` for Qwen3-8B Base (total_n=1000; missing samples are assigned 0).

J Failure Analysis: Prompt-Evidence Visibility Audit

Setup. In our training data construction (Sec. 4.2.2), we apply a strict Visibility Check: any identifier used by a step must be grounded in the model-visible context at that step, and we discard the entire oracle trajectory if any step violates visibility. This is motivated by partial observability induced by context control, where earlier observations can be deterministically replaced by placeholders to stay within the context budget (Sec. 4.1). At evaluation time, however, the environment only enforces execution-time reference validation (e.g., MID visibility and existing set IDs) and *does not* require relation/attribute strings to be visible in the context; they are checked only for schema validity and backend executability.

To quantify how often a model "overreaches" beyond the prompt-visible candidates, we perform a post-hoc audit on a CWQ-dev subset (N=1000 requested) using Hit@1 with strict termination (`require_finished=True`). We mark an episode as EVIDENCE-FAIL if *any* executed step uses an argument that is not visible in the step prompt under the same context truncation strategy (i.e., the SFT-style visibility constraint). We then report a *visibility-checked* Hit@1 (VC) that discounts hits with any evidence-fail.

Overall impact. On this subset, the standard Hit@1 is 74.0%. Evidence-fail occurs in 9.9% of episodes, but only 0.9% are hits that would be "blocked" by strict evidence gating, yielding a visibility-checked Hit@1 of 73.1% (74.0−0.9). Notably, evidence-fail is a strong signal of failure: conditioning on *no* evidence-fail, Hit@1 is 80.8% (728/901), while conditioning on evidence-fail, Hit@1 drops to 9.1% (9/99).

Among all errors (26.3%), 9.0% (90/1000) are errors that also trigger evidence-fail, i.e., 34.2% of the errors (90/263) are associated with prompt-visibility violations.

First-fail attribution on incorrect episodes. Table 15 breaks down the *first* evidence-fail reason on incorrect episodes (Hit@1=0). The dominant category is `hop_rel_not_visible` (36.7%), indicating that the model frequently selects a relation not present in the visible candidate list (mostly on ForwardHop). The next largest categories are `nodefeature_id_not_visible` (23.3%) and `nodefeature_attr_not_visible` (13.3%), suggesting that when the model fails, it often queries node features on IDs or attributes that are not supported by prompt-visible evidence. Evidence-fail for set operators is also non-trivial: `setop_attr_not_visible` (11.1%) and `setop_set_not_visible` (10.0%) typically correspond to filtering/sorting on unseen keys or referencing set IDs not present in the rendered set registry (e.g., due to token-level truncation after compression).

"Correct but blocked" cases (false positives under strict evidence gating). We find 9 episodes (0.9%) that are correct under standard evaluation (Hit@1=1) but would be blocked by strict evidence gating: ForwardHop+`hop_rel_not_visible` (4), NodeFeature+`nodefeature_attr_not_visible`

First evidence-fail reason (Hit@1=0)	#	% of 90
hop_rel_not_visible	33	36.7
nodefeature_id_not_visible	21	23.3
nodefeature_attr_not_visible	12	13.3
setop_attr_not_visible	10	11.1
setop_set_not_visible	9	10.0
hop_src_not_visible	4	4.4
nodefeature_missing_attr	1	1.1

Table 15: First-fail attribution among incorrect episodes that trigger prompt-evidence violations (90 episodes).

(2), FILTER+setop_attr_not_visible (2), and NODEFEATURE+nodefeature_id_not_visible (1). These cases typically arise from a mismatch between *executability* and *prompt visibility*: the model executes a correct KB operation, but our visibility definition is stricter and requires that the chosen relation/attribute/set ID appears in the step's rendered candidate lists. In practice, this can happen when (i) long predicate/attribute inventories are truncated in the displayed candidates, or (ii) an observation that would expose the needed candidates is compressed into a placeholder under context control.

Takeaway. Even under this strict audit (treating any evidence-fail as blocked), the drop from standard Hit@1 to visibility-checked Hit@1 is small (0.9 points). Meanwhile, evidence-fail strongly correlates with incorrectness, suggesting that prompt-evidence violations mostly reflect genuine "overreach" behavior rather than frequent false blocking.

K Filtering Coverage and Distributional Impact

We include a short example trace illustrating the set registry and evidence grounding (omitted here for brevity).

We also report the coverage and distributional impact of filtering. On CWQ, we compile expert SPARQL into linear tool plans for 99.8% of questions, and 74.4% admit visibility-grounded oracle rollouts under default resource limits (Table 1); importantly, the filter is applied only during SFT dataset construction and we do not discard instances at evaluation time.

Filtering induces a mild shift toward shorter plans: the filtered subset has lower mean plan length (5.989 vs. 6.610) and removes extremely long plans (max 16 vs. 34), which we disclose explicitly as a trade-off between executability and supervision coverage (Table 16).

Statistic	**Visibility Check–passed subset**	**CWQ train (full)**
Unique questions	20,643	27,639
Chat samples / lines	164,016	–
SQL2Plan success / parse error	20,643 / 0	27,585 / 54
Plan length mean	5.989	6.610
Plan length median	6	6
Plan length p90	9	10
Plan length max	16	34
$\Pr(\text{plan\_len} \geq 10)$	5.97%	12.86%
$\Pr(\text{plan\_len} \geq 11)$	2.32%	8.08%
$\Pr(\text{plan\_len} > 16)$	0.00%	> 0 (max=34)

Table 16: Plan-length distribution comparison under the same SQL2Plan configuration (with TOT_PLAN_COLLAPSE_HELPERS=1 and TOT_PLAN_SKIP_DIFF=1), where $\text{plan\_len} = |\text{stage\_linear\_steps}|$. Visibility Check filtering yields a subset with systematically shorter plans and removes extremely long plans.

T	**Hit@1$_{\text{FoF}}$ (%)**	T	**Hit@1$_{\text{FoF}}$ (%)**
1	0.0	9	61.9
2	0.0	10	68.5
3	0.0	11	71.6
4	0.0	12	73.3
5	15.9	13	73.7
6	30.5	14	73.7
7	36.8	15	73.7
8	47.4		

Table 17: Hit@1$_{\text{FoF}}$ under an action-step budget (counting all actions: Retrieve/Hop/Setop/NodeFeature/Finish). Evaluation is **finish-or-fail** (no forced Finish). $N = 1000$ requested, $N = 993$ loaded.

L Global Step Budget Sweep

Second, sweeping the global step budget T (counting all tool calls) shows a similar "ramp then plateau" behavior: Hit@1$_{\text{FoF}}$ is near zero for very small T, rises steadily as more actions are allowed, and saturates at 73.7 for $T \geq 13$ (Table 17).

Together, these results suggest that CWQ multi-hop execution typically requires a moderate number of tool calls beyond hop traversals alone, and that the primary bottleneck for base models is not insufficient budget but failure to use budget effectively under partial observability.